ECE-6483 Real Time Embedded Systems Homework 3

New York University, Spring 2025

Due on 04/22, 11:59 PM

Name: Raman Kumar Jha
NYU ID: N13866145

You are encouraged to work in groups, and to use the Internet or any other tools available to learn the material in order to answer these questions.

1. The ATmega128 microcontroller includes a UART that can be used to provide a serial interface. The following code snippet is often seen in programs that use the UART interface:

```
while(!(UCSR0A & 0x20));
UDR0 = x;
```

where x is a previously declared and initialized `uint8_t`; UCSR0A and UDR0 are defined in header files to refer to memory locations corresponding to the USART Control and Status Register A and USART Data Register, respectively; and the UART interface has already been configured, i.e. is ready for use.

(a) Refer to page 188 of the manual for the ATmega128, (available online). What does each line of the code snippet above do, with respect to the peripheral registers? What does the code snippet as a whole do?

Solution:

```
while(!(UCSR0A & 0x20));
UDR0 = x;
```

- The line `while(!(UCSR0A & 0x20));` waits until the USART Data Register Empty flag (UDRE0, bit 5 of UCSR0A) is set to 1, indicating that the transmit buffer is empty and ready to accept new data.

- The line `UDR0 = x;` writes the value of `x` (a previously defined `uint8_t`) into the USART Data Register, which initiates transmission of the byte over the serial interface.
- **As a whole:** This code snippet waits for the transmit buffer to become empty, then sends a byte of data over the UART interface.

(b) Suppose that the serial port operates at 57600 baud and the processor operates at 8 MHz. Approximately how many processor cycles are consumed by the code snippet above?

Solution:

$$\text{Time to send 8 bits} = \frac{8}{57600} \text{ seconds} \approx 138.9 \ \mu\text{s}$$

(c) To receive a byte over the serial port, a programmer might use the following code snippet to implement a readByte() function:

```
uint8_t readByte() {
while(!(UCSR0A & 0x80));
return UDR0;
}
```

What will happen if readByte() is called and there is no incoming byte over the serial interface?

Solution:

- If the transmit buffer is already empty, the while loop exits immediately, and only a few cycles are used.
- If the buffer is not empty, it takes 138.9 μs to send 8 bits. With a processor frequency of 8 MHz:

$$\text{Processor cycles} = 138.9 \times 10^{-6} \text{ s} \times 8 \times 10^{6} \text{ cycles/s} \approx 1112 \text{ cycles}$$

- During this time, the processor repeatedly checks the flag, doing no useful work.

(d) We say that a call to an I/O function is blocking if it blocks the calling program from continuing until the communication has finished. (Look up "Asynchronous I/O" on Wikipedia for more details.) Is a call to readByte() blocking? Why might this be problematic in some cases? Can you implement a non-blocking version of readByte()?

Solution:

```
uint8_t readByte() {
    while(!(UCSR0A & 0x80));
    return UDR0;
}
```

- If `readByte()` is called and there is no incoming byte, the function will block indefinitely, waiting for a byte to arrive. The calling program will not proceed until a byte is received.

(e) On this microcontroller, the baud rate is set by writing the value $UBRR = \frac{f_{osc}}{16 \cdot B_{des}} - 1$ to a UBRR register, where f_{osc} is the oscillator frequency in Hz and B_{des} is the desired baud rate in bits per second. The achieved baud rate is then $B_{ach} = \frac{f_{osc}}{16(UBRR+1)}$ (See page 172-173 of the ATmega128 reference manual for more details.) Because we can only write integer values to the register, not all baud rates can be achieved exactly.

- What is the closest we can get to 57600 baud (i.e., what is B_{ach}) if f_{osc} is 8 MHz? (Assume U2X is 0.)
- What value should be written to the UBRR register to achieve this baud rate?
- What is the percent error in this case, calculated as $\left(\frac{B_{\text{ach}}}{B_{\text{des}}} - 1\right) \times 100\%$

Solution:
Baud rate calculation for $f_{osc} = 8$ MHz, $B_{des} = 57600$ baud, $U2X = 0$:

- The UBRR register value is:

$$\text{UBRR} = \frac{f_{osc}}{16 \times B_{des}} - 1$$

- Plug in values:

$$\text{UBRR} = \frac{8,000,000}{16 \times 57600} - 1 = \frac{8,000,000}{921,600} - 1 \approx 7.68$$

- Closest integer is 8.
- Achieved baud rate:

$$B_{ach} = \frac{8,000,000}{16 \times (8+1)} = \frac{8,000,000}{144} \approx 55,555 \text{ baud}$$

- Percent error:

$$\text{Percent error} = \left|\frac{55,555}{57,600} - 1\right| \times 100\% \approx 3.56\%$$

(f) Suppose the other communication partner is an ATmega128 using $f_{osc} = 2$MHz. (Assume U2X is 0.) What will its B_{ach} be if it tries to operate at 57600 baud? What will be the total error between the pair, and is it less than the maximum error recommended in Table 75 of the ATmega128 reference manual (page 186)?

Solution

Baud rate for $f_{osc} = 2$ MHz, $B_{des} = 57600$ baud, $U2X = 0$:

- UBRR calculation:

$$\text{UBRR} = \frac{2,000,000}{16 \times 57600} - 1 = \frac{2,000,000}{921,600} - 1 \approx 1.17$$

- Closest integer is 1.
- Achieved baud rate:

$$B_{ach} = \frac{2,000,000}{16 \times (1+1)} = \frac{2,000,000}{32} = 62,500 \text{ baud}$$

- Percent error:

$$\text{Percent error} = \left| \frac{62,500}{57,600} - 1 \right| \times 100\% \approx 8.5\%$$

- If two devices communicate, one at $62,500$ baud, the other at $55,555$ baud, the total error is:

$$\text{Total error} = \left| \frac{62,500 - 55,555}{57,600} \right| \times 100\% \approx 12.1\%$$

- This is higher than the recommended maximum error (usually 2%).

2. Assume you have four (slave) devices connected to a (master) microcontroller over a shared I2C bus that uses standard (7-bit) addressing and is running at 400 kHz (most I2C devices can communicate at 100 kHz or 400 kHz).

(a) Draw a connection diagram for this configuration. What is the total number of wires?

Solution:

For an I2C bus with one master microcontroller and four slave devices, we need:

- Two signal wires shared among all devices:
  - SDA (Serial Data Line)

  - SCL (Serial Clock Line)

- Power connections (VCC and GND) for each device
- Pull-up resistors on both SDA and SCL lines

The connection diagram would show:

- One master microcontroller with SDA and SCL pins
- Four slave devices, each with SDA and SCL pins
- All SDA pins connected to a single SDA line with one pull-up resistor
- All SCL pins connected to a single SCL line with one pull-up resistor

Total number of wires: 2 wires (SDA and SCL)

This is one of the advantages of I2C - it requires only two wires regardless of how many devices are connected to the bus.

(b) Suppose the microcontroller reads one data byte from each of the four devices sequentially. (This is similar to the single-byte read shown on page 33 of the lecture slides, but instead of a stop at the end, there is a repeated start condition followed by a different slave address.) What is the data transfer rate in (data) bits per second from each device? (In other words, over some long interval of time T, how many bytes can slave S_i transmit?)

Solution:

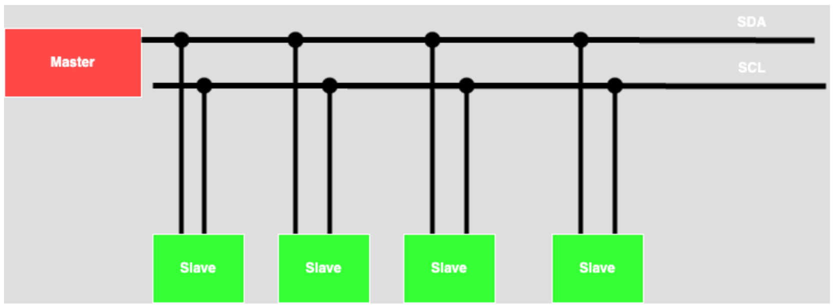

Figure 1: Connection diagram for I2C

For I2C communication with sequential single-byte reads from four devices: Each byte transfer involves:

- Start condition
- 7-bit slave address + 1 R/W bit
- 1 acknowledgment bit

- 8 data bits
- 1 acknowledgment bit
- Repeated start condition for next device (or stop condition for the last device)

For a single device read:

- Start condition: 1 bit time
- Address + R/W: 8 bit times
- Acknowledgment: 1 bit time
- Data byte: 8 bit times
- Acknowledgment: 1 bit time
- Repeated start/Stop: 1 bit time

Total: 20 bit times per device read
For all four devices: $4 \times 20 = 80$ bit times
At 400 kHz clock rate, the time for one bit is $\frac{1}{400{,}000} = 2.5\mu s$
Total time for reading from all four devices: $80 \times 2.5\mu s = 200\mu s$
Data transfer rate for each device:

$$\text{Data rate} = \frac{8 \text{ bits}}{200\mu s} \times \frac{1}{4} \qquad = \frac{8 \text{ bits}}{800\mu s} = 10,000 \text{ bits/second} = 10 \text{ kbps} \tag{1}$$

Over a long interval T, slave S_i can transmit:

$$\text{Bytes transmitted} = \frac{T \times 10,000 \text{ bits/s}}{8 \text{ bits/byte}} \qquad = T \times 1,250 \text{ bytes/s} \tag{2}$$

Therefore, each slave device can transmit at a rate of 1,250 bytes per second, or equivalently, 10 kbps.

(c) Repeat parts (a) and (b) for the SPI equivalent of the same setup.

Solution:
For SPI communication with one master and four slave devices:
Connection diagram:

- MOSI (Master Out Slave In) - 1 line connected to all slaves
- MISO (Master In Slave Out) - 1 line connected to all slaves
- SCLK (Serial Clock) - 1 line connected to all slaves
- SS (Slave Select) - 4 separate lines, one for each slave

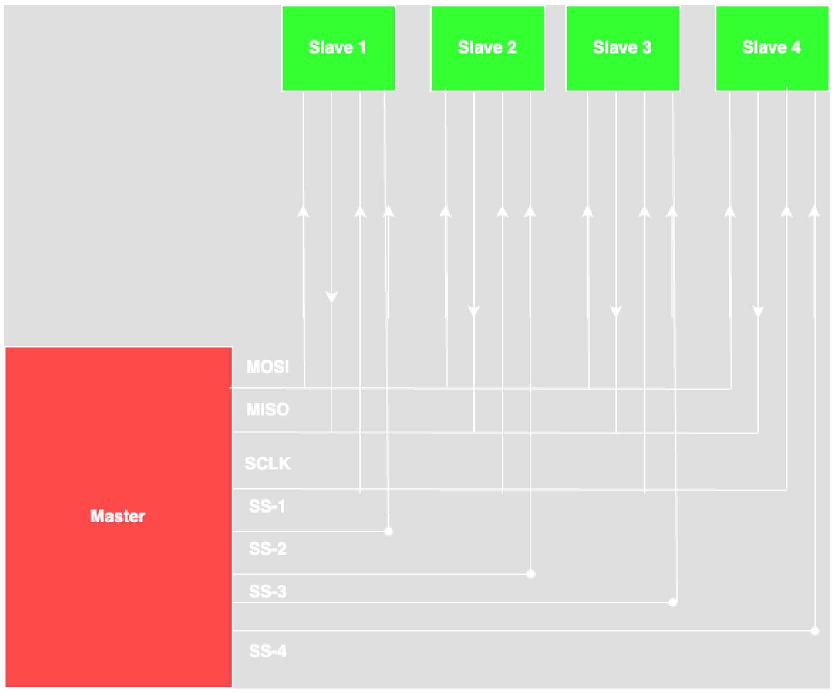

Figure 2: Connection diagram for SPI

Total number of wires: 3 shared lines (MOSI, MISO, SCLK) + 4 individual SS lines = 7 wires

Data transfer rate calculation:

For SPI, data transfer is simpler than I2C:

- Activate the SS line for the target slave
- Send 8 clock pulses to transfer 8 bits
- Deactivate the SS line

For a single device read at 400 kHz:

- SS activation/deactivation: negligible time
- 8 clock cycles at 400 kHz: $\frac{8}{400{,}000} = 20\mu s$

For all four devices: $4 \times 20\mu s = 80\mu s$

Data transfer rate for each device:

$$\text{Data rate} = \frac{8 \text{ bits}}{80\mu s} \times \frac{1}{4} \qquad = \frac{8 \text{ bits}}{320\mu s} = 25,000 \text{ bits/second} = 25 \text{ kbps} \tag{3}$$

Over a long interval T, slave S_i can transmit:

$$\text{Bytes transmitted} = \frac{T \times 25,000 \text{ bits/s}}{8 \text{ bits/byte}} = T \times 3,125 \text{ bytes/s} \tag{4}$$

Therefore, each slave device can transmit at a rate of 3,125 bytes per second, or equivalently, 25 kbps.

SPI achieves a higher data rate than I2C for the same clock frequency because it has less overhead (no addressing or acknowledgment bits).